tutorial - 5

NAME - DIVYANSH DUBEY
SECTION - F
ROLL No - 55
UNIVERSITY ROLL No - 2016738

Q What is the difference between DFS and BFS. Write applications of both the algorithms.

Ans

BFS	DFS
1) It stands for Breadth first search.	1) It stands for Depth first search.
2) It uses queue data structure.	2) It uses stack data structure.
3) It is more suitable for searching vertices which are clear to given source.	3) It is more suitable when there are solutions away from source.
4) It considers all neighbours first and therefore not suitable for decision making trees used in games & puzzles.	4) It is more suitable for game or puzzle problems. We make a decisio and then explore all paths.
5) Here siblings are visited before children.	5) Here children are visited befo siblings.
6) There is no concept of backtracking.	6) It is recursive algorithm tha uses backtracking.
7) It requires more memory.	7) It requires less memory.

Applications :-

BFS → Bipartite graph and shortest path, peer to peer networking, crawlers in search engine and GPS navigation system

DFS → acyclic graph, topological order, scheduling problems, sudoku puzzle.

Q2 Which data structure is used to implement BFS and DF
and why?

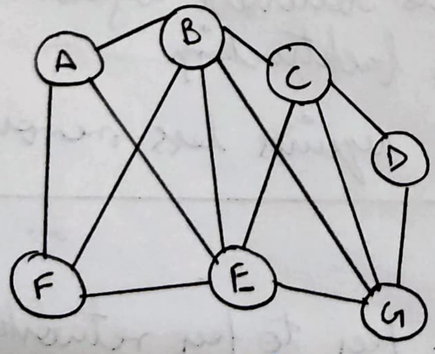
Ans For implementing BFS we need a queue data structure for finding shortest path between any node. We use queue bc things don't have to processed immediately, but have to be processed in FIFO order likes BFS. It searches for nodes level wise i.e it searches nodes w.r.t their distance from root. For this queue is better to use in BFS.

For implementing DFS we need a stack data structure as it transverses a graph in depthword motion and uses stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

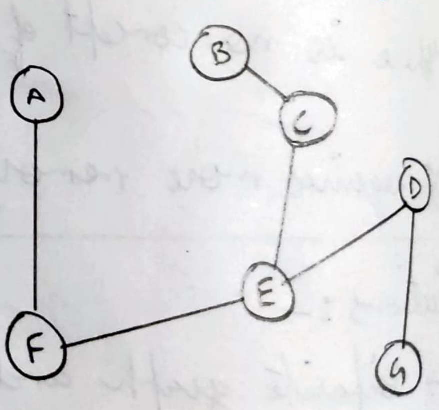
Q3 What do you mean by sparse and dense graphs? Which representation of graph is better for sparse and dense graphs

Ans Dense graph is a graph in which no. of edges is close to maximal no. of edges.

Sparse graph is graph in which no. of edges is very less.



Dense graph



Sparse graph

→ For sparse graph it is preffered to use adjacent list

→ For dense graph it is preffered to use adjacency matrix.

Q4 How can you detect a cycle in graph using BFS and DFS.

Ans For detecting cycle we need to use Kahn's algorithm for Topological sorting.

The steps involved are-

1) Compute in-degree (no. of incoming edges) for each of vertex present in graph and initialise count of visited nodes as 0.

2) Pick all vertices with in-degree as 0 and add them in queu

3) Remove a vertex from queue and then
→ Increment count of visited nodes as 1.
→ Decrease in-degree by 1 for all its neighbouring nodes.
→ If in-degree of neighbouring nodes is reduced to zero then add to queue.

4) Repeat 3 until queue is empty.

5) If count of visited nodes is not equal to no. of nodes in graph it has cycle otherwise not.

For detecting cycle in graph using DFS we need to do the following-

DFS for a connected graph produces a tree. There is cycle in graph if there is a back edge present in the graph. A back edge is an edge that is from a node to itself o one of its ancestored produced in the tree by DFS. To detect a back edge keep track of vertices currently in recu track for DFS transversal.

Q5 What do you mean by disjoint set data structure? Exp
3 operations along with examples which can be perform
on disjoint sets.

Ans A disjoint set is a data structure that keeps track of
set of elements partitioned into several disjoint sub-sets.
In other words, a disjoint set is a group of sets whe
no item can be in more than one set.

3 Operations

Find- Can be implemented by recursively transversing the p
array until we hit a node which is parent ~~of~~ to itse

eg
```
        int find (int i)
            if ( parent [i] == i) {
                return i;
            }
            else {
          return find ( parent [i]);
          }
        }
```

Union- It takes two elements as input. And find representati
of their sets using the find operation and finally puts e
one of the trees under root node of other tree, effectivly
merging the trees and sets.

eg
```
            Void union (int i, int j){
                int irep = this . Find (i);
                int jrep = this. Find (j);
```

Union by rank - We need a new array rank [ ]. Size of array same as parent array. If i is representative of set, rank [i] is height of tree. We need to minimise height of tree, if we are uniting two trees we call them left and right, then it all depends ~~of~~ on ranks of left and right.

. If rank of left is less than right then its best to move left under right and vice-versa.

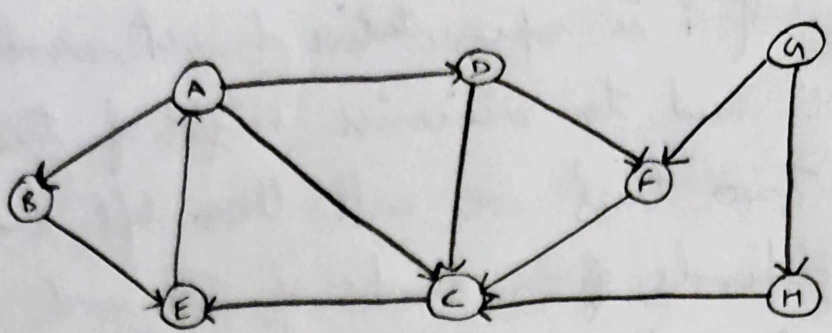
. If ranks are equal, ranks of result will always be one greater than rank of trees.

eg-

```
Void union (int i, int j) {
  int irep = this. Find (i);
  int jrep = this. Find (j);
      if (irep == jrep) return;
        irank = Rank [irep];


  jrank = Rank [jrep]
   if (irank < jrank
      this. parent [irep] = jrep;
   else if (jrank < irank)
     this. parent [jrep] = irep;
      else {
  this. parent [irep] = jrep;
     Rank [jrep] ++;
   }
}
```

Q6 Run BFS and DFS on graph shown below.



BFS

Child	G	H	D	F	C	E	A	B
Parent		G	G	G	H	C	E	A

Path → G → H → C → E → A → B

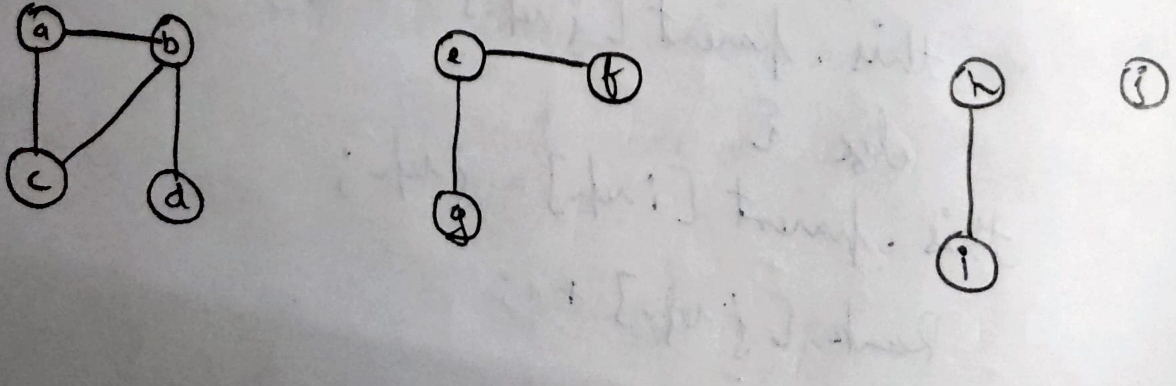
DFS

G
D
H
F
C
E
A
B
NODES VISITED

G
F
C
E
A
B
STACK

Path → G → F → C → E → A → B

Q7 Find out no. of connected components and vertices in each component using disjoint set data structure



V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

Ans E = {a,b}, {a,c}, {b,c}, {b,d}, {e,f}, {e,g}, {h,i}, {j}

(a,b) {a,b} {c} {d} {e} {f} {g} {h} {i} {j}

(a,c) {a,b,c} {d} {e} {f} {g} {h} {i} {j}

(b,c) {a,b,c} {d} {e} {f} {g} {h} {i} {j}

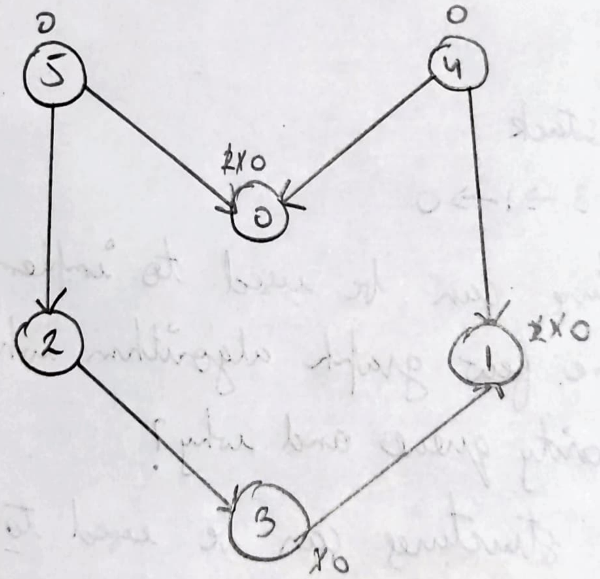
(b,d) {a,b,c,d} {e} {f} {g} {h} {i} {j}

(e,f) {a,b,c,d} {e,f} {g} {h} {i} {j}

(e,g) {a,b,c,d} {e,f,g} {h,i} {j}

No. of connected components = 3

Q8 Apply topological sort and DFS on graph having vertices from 0 to 5.



Ans We take ~~[illegible]~~ source node as 5. q : 5/4 ; Pop 5 and decrement indegree of by 1

Applying topological sort

$$\begin{tikzcd}
\text{DFS}(5) \arrow[d] \arrow[dd, bend right] \\
\text{DFS}(0) \arrow[u, bend right] \\
\text{DFS}(2) \arrow[d] \arrow[uu, bend right] \\
\text{DFS}(3) \arrow[d] \\
\text{DFS}(1)
\end{tikzcd}$$

q: 4/2 ; Pop 4 and decrement indegree and push 0

q: 2/0 Pop 2 and decrement indegree and push 3

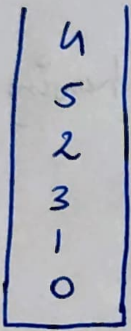
q: 0/3 Pop 0, Pop 3
Push 1

q: 1 ; Pop 1

Ans : 542031

Topological sort

DFS



stack

$4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 0$

Q9 Heap data structure can be used to implement priority queue. ~~Name~~ Name few graph algorithm where you need to use priority queue and why?

Ans Yes, heap data structures can be used to implement priority queue. It will take O(log N) time to inser delete each element in priority queue has two types priority queue based on max heap and minimum queue based on min heap. Heaps provide better per comparison to array.

The graphs like Dijkstra's shortest path algorithm, Prim's minimum spanning Tree use Priority Queue-

Dijkstra's Algorithm - When graph is stored in form of adjacency list or matrix, priority queue is used to extract minimum efficiently when implementing the algorithm.

Prim's Algorithm - It is used to store keys of nodes and extract minimum key node at every step

Q10 Differentiate between min-heap and max-heap

min Heap	max heap
1) In min heap, key present at root node must be less than or equal to among keys at all of its children.	1) In max heap the key present at root node must be greater than or equal to among keys present at all of its children.
2) minimum key element is present at root.	2) maximum key element pres at root.
3) It uses ascending priority	3) It uses descending priority
4) The smallest element has priority while construction of min heap	4) The largest element has priority while construction of ma
5) Smallest element first to be popped	5) largest element first to be popped